AD

AD-E403 689

Technical Report ARWSE-TR-14026

STD::STRING APPEND

Tom Nealis

October 2015

U.S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND ENGINEERING CENTER

Weapons and Software Engineering Center

Picatinny Arsenal, New Jersey

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

The citation in this report of the names of commercial firms or commercially available products or services does not constitute official endorsement by or approval of the U.S. Government.

Destroy this report when no longer needed by any method that will prevent disclosure of its contents or reconstruction of the document. Do not return to the originator.

UNCLASSIFIED

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-01-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (*From - To*)
October 2015	Final	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
STD::STRING APPEND	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHORS	5d. PROJECT NUMBER
Tom Nealis	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army ARDEC, WSEC Fire Control Systems & Technology Directorate (RDAR-WSF-M) Picatinny Arsenal, NJ 07806-5000	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
U.S. Army ARDEC, ESIC Knowledge & Process Management (RDAR-EIK) Picatinny Arsenal, NJ 07806-5000	11. SPONSOR/MONITOR'S REPORT NUMBER(S) Technical Report ARWSE-TR-14026

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Appending two or more strings together while developing a C++ application is a very common task. For std::strings, there are two primary ways to achieve the appended string. The first is to use the += operator to append two strings, and the second is to use the + operator. This report compares the two operations.

15. SUBJECT TERMS

std::string Append

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Tom Nealis
U	U	U	SAR	9	19b. TELEPHONE NUMBER (Include area code) (973) 724-8048

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

CONTENTS

INTRODUCTION

The C++ developers are well aware of how useful std::strings += and + operators are but not always aware of which one is more efficient and in turn better to use. The + operator may provide the coder the ability to put more code on a single line, but it turns out that the assembly code produced is far less efficient.

METHODOLOGY

In order to acquire data for this report, a program was written that would concatenate strings using the += operator and also concatenate the same two strings using the + operator. Data was collected for concatenating 2 to 10 strings. The source code for this program is shown in the following sequence:

```
int _tmain(int argc, _TCHAR* argv[])
{
 LARGE_INTEGER frequency;
 QueryPerformanceFrequency(&frequency);

 LARGE_INTEGER starting_time, ending_time, elapsed_microseconds;

 //std::ofstream a_file("outfile2.txt");
 //std::ofstream a_file("outfile3.txt");
 //std::ofstream a_file("outfile4.txt");
 //std::ofstream a_file("outfile5.txt");
 //std::ofstream a_file("outfile6.txt");
 //std::ofstream a_file("outfile7.txt");
 //std::ofstream a_file("outfile8.txt");
 //std::ofstream a_file("outfile9.txt");
 std::ofstream a_file("outfile10.txt");


 //setup strings here
 std::vector<std::string> my_strings;
 my_strings.push_back("This is the first.");
 my_strings.push_back("This is the second.");
 my_strings.push_back("This is the third.");
 my_strings.push_back("This is the fourth.");
 my_strings.push_back("This is the fifth.");
 my_strings.push_back("This is the sixth.");
 my_strings.push_back("This is the seventh.");
 my_strings.push_back("This is the eighth.");
 my_strings.push_back("This is the nineth.");
 my_strings.push_back("This is the tenth.");

 std::string plus_equal;
 std::string plus_plus;

 for(auto i = 0u; i < 10; ++i)
 {
  plus_equal = "";
  QueryPerformanceCounter(&starting_time);

  //code to measure here
  plus_equal = my_strings[0];
  plus_equal += my_strings[1];
  plus_equal += my_strings[2];
  plus_equal += my_strings[3];
  plus_equal += my_strings[4];
  plus_equal += my_strings[5];
  plus_equal += my_strings[6];
  plus_equal += my_strings[7];
  plus_equal += my_strings[8];
```

```
    plus_equal += my_strings[9];

    QueryPerformanceCounter(&ending_time);
    elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    //this time is in micro seconds
    auto te_plus_equal = static_cast<double>((elapsed_microseconds.QuadPart * 1000000.0) / frequency.QuadPart);


    plus_plus = "";
    QueryPerformanceCounter(&starting_time);

    //code to measure here
    //plus_plus = my_strings[0] + my_strings[1];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2] + my_strings[3];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2] + my_strings[3] + my_strings[4];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2] + my_strings[3] + my_strings[4] + my_strings[5];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2] + my_strings[3] + my_strings[4] + my_strings[5] + my_strings[6];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2] + my_strings[3] + my_strings[4] + my_strings[5] + my_strings[6] +
my_strings[7];
    //plus_plus = my_strings[0] + my_strings[1] + my_strings[2] + my_strings[3] + my_strings[4] + my_strings[5] + my_strings[6] +
my_strings[7] + my_strings[8];
    plus_plus = my_strings[0] + my_strings[1] + my_strings[2] + my_strings[3] + my_strings[4] + my_strings[5] + my_strings[6] +
my_strings[7] + my_strings[8] + my_strings[9];

    QueryPerformanceCounter(&ending_time);
    elapsed_microseconds.QuadPart = ending_time.QuadPart - starting_time.QuadPart;

    //this time is in micro seconds
    auto te_plus_plus = static_cast<double>((elapsed_microseconds.QuadPart * 1000000.0) / frequency.QuadPart);


    a_file << te_plus_equal << "," << te_plus_plus << "\r\n";


    printf("Run: %d \t\tte plus equal: %4.2f \t\tte plus plus: %4.2f\r\n", i + 1, te_plus_equal, te_plus_plus);
  }

  a_file.close();

  printf("All done!\n");

  //this stops the program in order to see data;
  getchar();

  return 0;
}
```

The code is very straightforward. Sections need to be commented out depending on the results that are desired. The built-in, high-resolution counters are used in order to measure how long a snippet of code took. The results are logged to the output file for later processing.

After running this program for each of the results desired, the results are shown in figure 1.

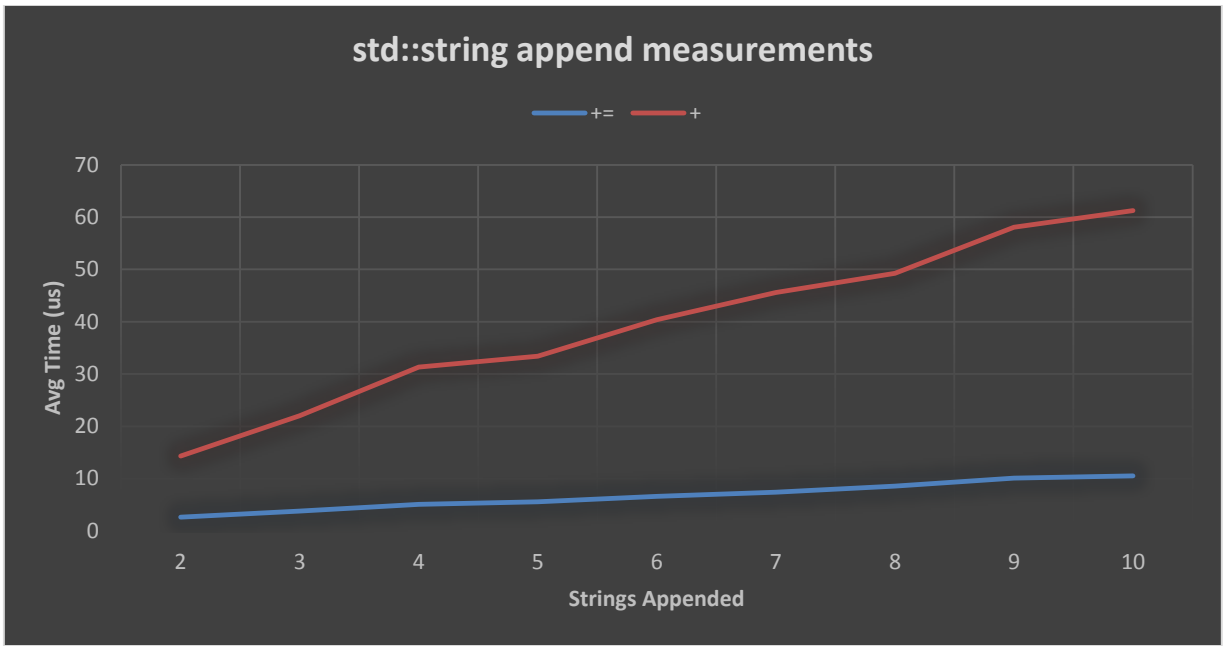

Figure 1
std::string append measurements

Figure 1 clearly shows that the time needed to append two std::strings using the + operator takes significantly longer than the += operator.

Let's take a look at look at the compiler generated assembly code in order to get a better idea of why the measured results were received. For appending three std::strings, the assembly code is as follows:

```
plus_equal = my_strings[0];
00C2E39A  push      0
00C2E39C  lea       ecx, [my_strings]
00C2E3A2  call      std::vector<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > > >::operator[] (0C21014h)
00C2E3A7  push      eax
00C2E3A8  lea       ecx, [plus_equal]
00C2E3AE  call      std::basic_string<char, std::char_traits<char>, std::allocator<char> >::operator= (0C212BCh)
plus_equal += my_strings[1];
00C2E3B3  push      1
plus_equal += my_strings[1];
00C2E3B5  lea       ecx, [my_strings]
00C2E3BB  call      std::vector<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > > >::operator[] (0C21014h)
00C2E3C0  push      eax
00C2E3C1  lea       ecx, [plus_equal]
00C2E3C7  call      std::basic_string<char, std::char_traits<char>, std::allocator<char> >::operator+= (0C21217h)
plus_equal += my_strings[2];
00C2E3CC  push      2
00C2E3CE  lea       ecx, [my_strings]
00C2E3D4  call      std::vector<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > > >::operator[] (0C21014h)
00C2E3D9  push      eax
00C2E3DA  lea       ecx, [plus_equal]
00C2E3E0  call      std::basic_string<char, std::char_traits<char>, std::allocator<char> >::operator+= (0C21217h)

18 instructions
```

```
plus_plus = my_strings[0] + my_strings[1] + my_strings[2];
00C2E467  push      2
00C2E469  lea       ecx, [my_strings]
00C2E46F  call      std::vector<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > > >::operator[] (0C21014h)
00C2E474  push      eax
00C2E475  push      1
00C2E477  lea       ecx, [my_strings]
00C2E47D  call      std::vector<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > > >::operator[] (0C21014h)
00C2E482  push      eax
00C2E483  push      0
00C2E485  lea       ecx, [my_strings]
00C2E48B  call      std::vector<std::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::allocator<std::basic_string<char,
std::char_traits<char>, std::allocator<char> > > >::operator[] (0C21014h)
00C2E490  push      eax
00C2E491  lea       eax, [ebp - 290h]
00C2E497  push      eax
00C2E498  call      std::operator+<char, std::char_traits<char>, std::allocator<char> > (0C216C7h)
00C2E49D  add       esp, 0Ch
00C2E4A0  mov       dword ptr[ebp - 40Ch], eax
00C2E4A6  mov       ecx, dword ptr[ebp - 40Ch]
00C2E4AC  mov       dword ptr[ebp - 410h], ecx
00C2E4B2  mov       byte ptr[ebp - 4], 0Eh
00C2E4B6  mov       edx, dword ptr[ebp - 410h]
00C2E4BC  push      edx
00C2E4BD  lea       eax, [ebp - 26Ch]
00C2E4C3  push      eax
00C2E4C4  call      std::operator+<char, std::char_traits<char>, std::allocator<char> > (0C211AEh)
00C2E4C9  add       esp, 0Ch
00C2E4CC  mov       dword ptr[ebp - 414h], eax
00C2E4D2  mov       ecx, dword ptr[ebp - 414h]
00C2E4D8  push      ecx
00C2E4D9  lea       ecx, [plus_plus]
00C2E4DF  call      std::basic_string<char, std::char_traits<char>, std::allocator<char> >::operator= (0C217A8h)
00C2E4E4  lea       ecx, [ebp - 26Ch]
00C2E4EA  call      std::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string<char, std::char_traits<char>,
std::allocator<char> >(0C2164Fh)
00C2E4EF  mov       byte ptr[ebp - 4], 0Dh
00C2E4F3  lea       ecx, [ebp - 290h]
00C2E4F9  call      std::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string<char, std::char_traits<char>,
std::allocator<char> >(0C2164Fh)

36 instructions
```

The += append only created 18 lines of machine code versus the 36 lines of machine code generated by the + operator. So just by the number of instructions created, one can see that the + operator will take longer. Looking deeper into the assembly, it can be seen that the + operator is returning a new buffer for each +, whereas the += operator is doing an actual concatenation.

CONCLUSIONS

It's very important for a developer to understand the complexities of writing code in one way versus another. This report clearly shows that the more efficient way to append std::strings is to use the += operator.

DISTRIBUTION LIST

U.S. Army ARDEC
ATTN: RDAR-EIK
RDAR-WSF-M, T. Nealis
Picatinny Arsenal, NJ 07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste 0944
Fort Belvoir, VA 22060-6218

GIDEP Operations Center
P.O. Box 8000
Corona, CA 91718-8000
gidep@gidep.org

REVIEW AND APPROVAL OF ARDEC TECHNICAL REPORTS

std::string Append
Title

Date received by LCSD

Thomas M. Nealis
Author/Project Engineer

Report number (to be assigned by LCSD)

X8048 31 RDAR-WSF-M

Extension Building

Author's/Project Engineers Office
(Division, Laboratory, Symbol)

PART 1. **Must be signed before the report can be edited.**

a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.

b. Use Distribution Statement A_X___, B__, C___, D____, E____, F____ or X____ for the reason checked on the continuation of this form. Reason: ____Operational Use_____

1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.

2. If Statement B, C, D, E, F, or X is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.

c. The distribution list for this report has been reviewed for accuracy and completeness.

Patricia Alameda 9/9/15
Division Chief (Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

Patricia Alameda 9/9/15
Division Chief (Date)

Andrew Pskowski 9/15/15
RDAR-CIS (Date)

LCSD 49 supersedes SMCAR Form 49, 20 Dec 06